2de3	DM n°3 : Programmation/Équation

Au choix : tout faire à deux, ou bien travailler seul en ne traitant qu'une des trois questions de la partie I.

I] Fonction zêta

La fonction ζ (zêta) de Riemann joue un rôle important en théorie des nombres.

L'image d'un entier n par ζ est égal à $\frac{1}{1^n}+\frac{1}{2^n}+\frac{1}{3^n}+...$ (jusqu'à l'infini).

D'une façon générale, Euler prouva que cette somme infinie était égale au produit infini $(\frac{2^n}{2^n-1})\times(\frac{3^n}{3^n-1})\times(\frac{5^n}{5^n-1})...$ où 2, 3, 5, etc. sont les nombres premiers.

a) ζ(1) : La série harmonique

La somme des inverses des entiers positifs, de 1 à l'infini, est notée ζ(1). Pour un entier positif n notons : $F(n)=\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+...+\frac{1}{n}$ et $F'(n)=\frac{2}{1}\times\frac{3}{2}\times\frac{5}{4}...\times\frac{p}{p-1}$ où p le $n^{\text{ème}}$ nombre premier (divisible uniquement par 1 et par p).

Ainsi $F(1)=\frac{1}{1}=1$, $F(2)=\frac{1}{1}+\frac{1}{2}=\frac{3}{2}$, ... et $F'(1)=\frac{2}{1}=2$, $F'(2)=\frac{2}{1}\times\frac{3}{2}=3$, ...

►Déterminer les valeurs exactes de $F(n)$ et de $F'(n)$ pour $n\in\{3,4,5,6,7\}$.

$F_3=\frac{1}{1}+\frac{1}{2}+\frac{1}{3}=\frac{3}{2}+\frac{1}{3}=\frac{11}{6}\approx 1{,}833$	$F'(3)=\frac{2}{1}\times\frac{3}{2}\times\frac{5}{4}=\frac{30}{8}=3{,}75$
$F_4=\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}=\frac{11}{6}+\frac{1}{4}=\frac{25}{12}\approx 2{,}083$	$F'(4)=\frac{2}{1}\times\frac{3}{2}\times\frac{5}{4}\times\frac{7}{6}=\frac{210}{48}=4{,}375$
$F_5=\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}=\frac{25}{12}+\frac{1}{5}=\frac{125+12}{60}=\frac{137}{60}\approx 2{,}283$	$F'(5)=\frac{2}{1}\times\frac{3}{2}\times\frac{5}{4}\times\frac{7}{6}\times\frac{11}{10}=\frac{2310}{480}\approx 4{,}81$
$F_6=\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\frac{1}{6}=\frac{137}{60}+\frac{1}{6}=\frac{147}{60}=2{,}45$	$F'(6)=\frac{2}{1}\times\frac{3}{2}\times\frac{5}{4}\times\frac{7}{6}\times\frac{11}{10}\times\frac{13}{12}=\frac{30030}{5760}\approx 5{,}21$
$F_7=\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\frac{1}{6}+\frac{1}{7}=\frac{147}{60}+\frac{1}{7}=\frac{1089}{420}\approx 2{,}593$	$F'(7)=\frac{2}{1}\times\frac{3}{2}\times\frac{5}{4}\times\frac{7}{6}\times\frac{11}{10}\times\frac{13}{12}\times\frac{17}{16}=\frac{510510}{92160}\approx 5{,}54$

Comparer ces deux séries de valeurs entre elles :

Progressent-elles de la même manière ? L'écart entre $F(n)$ et $F'(n)$ diminue t-il lorsque n croit ?

Ces deux séries de valeurs sont croissantes, la série F' ayant des valeurs plus importantes. Pour avoir quasi-égalité entre deux termes des deux séries, il faut prendre F_4 et F'_1 qui valent à peu près 2, ou après F_{12} et F'_2 qui valent à peu près 3...

Calculons les écarts entre $F(n)$ et $F'(n)$, ainsi que l'augmentation de ces écarts (en %).

n	1	2	3	4	5	6	7
$F'(n)-F(n)$	1	1,5	1,92	2,29	2,53	2,76	2,95
Augmentation		50%	28%	19%	10%	9%	7%
$F'(n)\div F(n)$	2	2	2,05	2,1	2,11	2,13	2,14

On voit que l'écart entre ces deux valeurs, même s'il se maintient, augmente de moins en moins. On peut imaginer que, comme les deux sommes infinies sont égales, les différences vont continuer à s'estomper progressivement.

►Écrire un algorithme qui permet de déterminer une valeur approchée de $F(n)$ quand on entre n.

Programmer cet algorithme et déterminer les valeurs approchées de $F(100)$, $F(1000)$ et $F(10000)$.

Ce n'est pas si facile de disposer de la liste de nombres premiers. Pour cette raison, nous ne nous intéresserons plus qu'à la suite des valeurs $F(n)$. Pour en calculer une valeur approchée à partir de n, voici l'algorithme utilisé :

Lire le nombre n
S=0
Pour i allant de 1 à n : S=S+1/i
Afficher S

Je programme sur mon ordinateur cet algorithme en langage Python :

```
n=int(input("Jusqu'à quelle rang? "))
S=0
for i in range(1,n+1):
    S+=1/i
print("Somme= {} ".format(S))
```

```
Jusqu'à quelle rang? 7
Somme= 2.5928571428571425
```

Je teste ensuite pour n=7 et trouve 2,59285714 ce qui me confirme l'exactitude de mon programme.

Déterminons maintenant S pour les valeurs indiquées :

n	10	100	1000	10000	100000
S	2.9289682539682538	5.187377517639621	7.485470860550343	9.787606036044348	12.090146129863335

Aucun problème. Sur la calculatrice, cela peut être un peu plus long, voire beaucoup plus long.

►Modifier l'algorithme pour qu'il détermine à partir de quelle valeur de n, $F(n)$ dépasse une valeur m fixée à l'avance. Programmer cela et déterminer la valeur de n pour laquelle $F(n)>10^6$.
D'après vos observations, que pensez-vous de ζ(1), c'est-à-dire de $F(n)$ quand n se rapproche de l'infini ?
On a constaté à la question précédente que $F(n)$ augmente sans montrer de signes d'essoufflement. On peut penser alors que cette suite de nombre va dépasser toutes les valeurs positives m fixée à l'avance. En d'autres termes cette suite se rapprocherait de l'infini quand n se rapproche de l'infini.
L'algorithme modifié qui va permettre de tester cette hypothèse est le suivant :

Lire le nombre m
S=0
i=1
Tant que S<m: S=S+1/i ; i=i+1
Afficher i et puis S

Nous allons tester le programme qui en résulte sur des valeurs progressivement de plus en plus grandes de m, pour voir si on arrive à dépasser ces valeurs avec la suite $F(n)$.

```
m=int(input("Jusqu'à quelle valeur? "))
S=0
i=1
while S<m:
    S+=1/i
    i+=1
print("Rang mini = {} - Somme= {} ".format(i,S))
```

```
Jusqu'à quelle valeur? 10
Rang mini = 12368 - Somme= 10.000043008275778

Jusqu'à quelle valeur? 20
Rang mini = 272400601 - Somme= 20.000000001618233
```

Le problème, c'est que la croissance est de plus en plus lente : pour dépasser 10 il faut 12 368 termes et pour dépasser 20 il en faut 272 400 601, soit 22 000 fois plus ! Cela semble donc impossible d'atteindre 1 000 000 et pourtant la théorie mathématique affirme que cette suite dépasse n'importe quelle valeur au bout d'un certain temps. Le véritable problème n'est même pas le temps de calcul. Il s'agit d'une limitation de la notation flottante des nombres réels : les nombres ajoutés deviennent trop petits pour modifier la somme S. Il y a un dépassement de la capacité pour mémoriser un nombre constitué d'autant de chiffres. On pourrait attendre jusqu'à la fin des temps, le nombre m=1 000 000 ne serait toujours pas dépassé... Désolé d'avoir posé une question dont la réponse ne tombe pas sous le sens.

m	10	20	30	100	100000
Premier rang où $F(n)>m$	12 368	272 400 601	?	??	???

Pour me faire pardonner, je vais écrire un 3ème programme qui donne le rang du terme qui est égal (pour l'ordinateur) au terme précédent. Je garde en mémoire le terme précédent, que je mets dans la mémoire S1 et je calcule le nouveau dans la mémoire S2. Le programme s'arrête quand S1=S2.

```
from time import clock
S1=0
S2=1
i=2
temps=clock()
while S1<S2:
    S1=S2
    S2+=1/(i)
    i+=1
    if i%1000000==0: print ("Rang:{} - S: {} - Temps:{} ".format(i,S2,clock()-temps))
print("Rang égalité = {} - Somme= {}- Temps:{} ".format(i,S1,clock()-temps))
```

```
Rang:282000000 - S: 20.034633292031668 - Temps:199.76398449707895
Rang:283000000 - S: 20.038173118743135 - Temps:200.51668056993117

Rang:24901000000 - S: 24.515389465032168 - Temps:17999.772317082807
Rang:24902000000 - S: 24.51542962340514 - Temps:18000.50123522583
```

Ce programme tourne assez longtemps pour qu'on puisse s'impatienter. Pour cela j'ai demandé l'affichage d'un résultat provisoire tous les millions de valeurs. Cela permet de savoir que le programme tourne bien et continue à chercher ce qu'il est censé chercher. La trace de l'exécution montre qu'au bout de 3 minutes et 20, on en est à $F(283000000)$ ≈20,038. Au bout de 5 heures (5h=18 000 s), on n'a pas beaucoup avancé mais tout de même : $F(24902000000)$ ≈24,515. Plus de 25 milliards de termes ajoutés et on n'arrive même pas à 25... Pour atteindre 25, je dois laisser tourner le programme sur mon ordinateur pendant plus de 8 heures (31185 secondes), il y a alors environ 43 millions de valeurs calculées. On n'est plus tout-à-fait dans des délais raisonnables. Le résultat vaut peut-être l'effort pour l'atteindre, alors je laisse mon ordinateur allumé travailler en sourdine pour la Science. Au bout de 60 heures (21600 secondes), le programme arrive péniblement à 27, après avoir ajouté les 300 milliards premiers nombres entiers. Il faut dire que $\frac{1}{30000000000}\approx 3{,}3\times10^{-12}$, un nombre qui peut s'ajouter à S=27 en changeant la 12ème décimale... Il y a en tout 14 chiffres significatifs dans ce nombre. La capacité de stockage du type flottant de Python n'est pas encore dépassée.

```
Rang:298837000000 - S: 27.00037973461653 - Temps:215724.6212359087
Rang:298838000000 - S: 27.000383081272815 - Temps:215725.36500688913
Rang:298839000000 - S: 27.0003864279291 - Temps:215726.07472982362
```

Je peux difficilement résister à l'envie d'écrire ici le raisonnement trouvé sur une copie, qui provient de l'examen de ce tableau de progression de la fonction Z :

10^n	1	10	10^2	10^3	10^4	10^5	10^6	10^7
$F(10^n)$	1	2.928	5.187	7.485	9.787	12.090	14.392	16.695
$F(10^n)-F(10^{n-1})$		1,928	2,259	2,298	2,302	2,303	2,302	2,303

On constate que l'augmentation de $F(n)$ se stabilise, d'une certaine façon, sur 2,3 à chaque fois que n est multiplié par 10. Ainsi, on peut prévoir que $F(10^6)\approx 12{,}090+2{,}3=14{,}39$. Le problème, si cette propriété reste inchangée, revient à déterminer le nombre de fois qu'il faut ajouter 2,3 pour arriver à 1 000 000. Une estimation de ce nombre est obtenue en divisant 1 000 000 par 2,3 : on trouve 434 782. Ce qui conduit au nombre énorme de 10^{434782} , un nombre qui s'écrit avec 434 782 chiffres ! Il va sans dire que l'intention de déterminer à partir de quand on dépasse 1 000 000 est vouée à l'échec avec nos simples programmes.
Une autre approche est de considérer ce résultat : la somme $F(n)$ et $\log(n)$ – log, abréviation de *logarithme népérien*, est une fonction étudiée en classe de T^ale^ – forment une différence qui, quand n se rapproche de l'infini, tend vers un nombre appelé constante d'Euler (encore lui!) qui vaut environ 0,5772156649. Autrement dit, quand n est très grand, on a $F(n)\approx\log(n)+0{,}5772156649$. L'inéquation $F(n)>10^6$ s'écrit $\log(n)+0{,}5772156649>10^6$, qui s'écrit $\log(n)>10^6-0{,}5772156649\approx 10^6$. Cette inéquation a pour solution : $n>e^{10^6}$, soit $n>10^{\frac{10^6}{\log 10}}\approx 10^{434294}$.

```
from time import clock
from math import log
n=int(input("On part de quel rang? "))
S=0.57721566490153286+log(n)
temps=clock()
while S+1/n>S:
    S+=1/n
    n+=1
    if n%1000000==0: print ("Rang:{} - S: {} - Temps:{} ".format(n,S,clock()-temps))
print("Rang égalité = {} - Somme= {}- Temps:{} ".format(n,S,clock()-temps))
```

```
On part de quel rang? 281474970000000
Rang:281474971000000 - S: 33.84828031504329 - Temps:0.7941327107884423
Rang:281474972000000 - S: 33.84828032214872 - Temps:1.666059877320986
Rang:281474973000000 - S: 33.848280329254145 - Temps:2.4601791981275447
Rang:281474974000000 - S: 33.84828033635957 - Temps:3.276481032421163
Rang:281474975000000 - S: 33.848280343465 - Temps:4.0841081669516495
Rang:281474976000000 - S: 33.84828035057043 - Temps:4.878275691692989
Rang égalité = 281474976710657 - Somme= 33.84828035561995- Temps:5.444866
```

Pour en revenir au problème que je me suis mis en tête de résoudre : trouver le rang du terme qui est égal (pour l'ordinateur) au terme précédent. Je vais procéder par tâtonnements puisque la méthode directe est trop longue (60h c'est beaucoup! Et pour un résultat sans doute loin du compte). J'augmente d'une puissance de 10 à chaque essai, jusqu'à dépasser le nombre cherché pour 500000000000000. Je descend alors 400000000000000 trop grand, 300000000000000 trop grand, 200000000000000 ce n'est pas assez, je passe alors à 250000000000000, etc. Jusqu'à arriver à 281474970000000 qui aboutit en 5 secondes : le nombre cherché est 281 474 976 710 657 ou pas loin. Près de 281 474 milliards d'inverses d'entiers à ajouter pour en arriver à un arrêt de la progression de cette suite. Arrêt erroné car la suite continue jusqu'à l'infini, mais arrêt technique dû au type « flottant » des nombres employés. La somme S est alors égale à 33.84828035561995 et elle ne progressera plus, tant que l'on calculera avec des nombres flottants de cette simple précision (il existe des nombres flottants de double précision). Avec un rythme de 60h pour 300 milliards de nombres entiers, il faudrait attendre près de 6 ans et demi pour atteindre les 281 474 milliards de nombres... J'ai bien fait d'arrêter au bout de 60h.

b) ζ(2) : Le problème de Mengoli (appelé aussi problème de Bâle − Bâle étant la ville de naissance d'Euler)
En 1735, Euler a montré que ce problème posé par Mengoli en 1644 − à savoir « *quelle est la somme des inverses des carrés des entiers positifs ?* » − avait pour solution l'irrationnel $\frac{\pi^2}{6}$. Pour cela, il montra l'identité des vingt premières décimales de $\frac{\pi^2}{6}$ et de $G(n)$.
Notons $G(n)=\frac{1}{1^2}+\frac{1}{2^2}+\frac{1}{3^2}+\ldots+\frac{1}{n^2}$ la somme des n premiers inverses des carrés d'entiers positifs.
►Calculer à la main les valeurs exactes de $G(n)$, pour $n\in\{1,2,3,4,5\}$.
$G_1=\frac{1}{1^2}=1$
$G_2=\frac{1}{1^2}+\frac{1}{2^2}=\frac{5}{4}=1{,}25$
$G_3=\frac{1}{1^2}+\frac{1}{2^2}+\frac{1}{3^2}=\frac{5}{4}+\frac{1}{9}=\frac{45+4}{36}=\frac{49}{36}\approx 1{,}3611\ldots$
$G_4=G_3+\frac{1}{4^2}=\frac{1}{16}+\frac{49}{36}=\frac{205}{144}\approx 1{,}424$
$G_5=G_4+\frac{1}{5^2}=\frac{1}{25}+\frac{205}{144}=\frac{5269}{3600}\approx 1{,}464$.

► Écrire un algorithme qui permet de déterminer une valeur approchée de $G(n)$ quand on entre n.
Programmer cet algorithme et déterminer ces valeurs de $G(n)$ pour $n\in\{10^1, 10^2, 10^3, 10^4, 10^5\}$.
Pour calculer une valeur approchée de G_n à partir de n, voici l'algorithme utilisé :

Lire le nombre n
S=0
Pour i allant de 1 à n : S=S+1/($i\times i$)
Afficher S

Je programme sur mon ordinateur cet algorithme en langage Python :

```
n=int(input("Jusqu'à quelle rang? "))
S=0
for i in range(1,n+1):
    S+=1/(i*i)
print("Somme= {} ".format(S))
```

```
Jusqu'à quelle rang? 5
Somme= 1.4636111111111112
```

Je teste ensuite pour n=5 et trouve 1,46361... ce qui me confirme l'exactitude de mon programme.

Déterminons maintenant s pour les valeurs indiquées :

n	10	100	1000	10000	100000
S	1.5497677311665408	1.6349839001848923	1.6439345666815615	1.6448340718480652	1.6449240668982423

Aucun problème. Sur la calculatrice, cela peut être un peu plus long, voire beaucoup plus long.

►Modifier l'algorithme pour qu'il détermine à partir de quelle valeur de n, $\frac{\pi^2}{6}-G(n)$ devient inférieur à une valeur $m=10^{-p}$ fixée à l'avance (p pour précision). Programmer et exécuter le programme pour $p\in\{1,2,3,4,5\}$. Pensez-vous qu'Euler calcula directement $G(n)$ avec une si grande précision ?
Avec cet algorithme, on ne détermine qu'une valeur approchée de G_n.
On peut évaluer l'écart avec la limite théorique : $\frac{\pi^2}{6}-G(1)\approx 0{,}645$; $\frac{\pi^2}{6}-G(2)\approx 0{,}395$; $\frac{\pi^2}{6}-G(3)\approx 0{,}284$; $\frac{\pi^2}{6}-G(4)=\frac{\pi^2}{6}-\frac{205}{144}\approx 0{,}221$; $\frac{\pi^2}{6}-G(5)=\frac{\pi^2}{6}-\frac{5269}{3600}\approx 0{,}181$; $\frac{\pi^2}{6}-G(100)\approx 0{,}01$; $\frac{\pi^2}{6}-G(1000)\approx 0{,}001$; $\frac{\pi^2}{6}-G(10000)\approx 0{,}0001$; $\frac{\pi^2}{6}-G(100000)\approx 0{,}00001$ ...
Il semblerait, au vu de ces résultats que $\frac{\pi^2}{6}-G(n)$ est de l'ordre de $\frac{1}{n}$ (pour n=100, on trouve $\frac{1}{100}$, pour n=1000, on trouve $\frac{1}{1000}$, etc.) et donc pour avoir 20 chiffres exacts, il faudrait aller jusqu'à calculer $G(10000000000000000000)$. Comme cela ne semble pas très raisonnable, on peut supposer qu'Euler ne calcula pas si loin (il ne disposait pas d'ordinateur...) Il développa des méthodes mathématiques plus sophistiquées que cette approche bêtement calculatoire que nous avons faite.
La question posée portait sur l'algorithme modifié :

Lire le nombre p
S=0
i=1
Tant que π²/6-S> 10^{-p} : S=S+1/($i\times i$) ; i=i+1
Afficher i et puis S

Voici le programme et ses exécutions en Python.

```
from math import pi

p=int(input("Jusqu'à quelle précision? "))
S=0
i=1
while pi**2/6-S>10**(-p):
    S+=1/(i*i)
    i+=1
print("Rang mini = {} - Somme= {} ".format(i,S))
```

```
Jusqu'à quelle précision? 1
Rang mini = 11 - Somme= 1.5497677311665408
Jusqu'à quelle précision? 2
Rang mini = 101 - Somme= 1.6349839001848923
Jusqu'à quelle précision? 3
Rang mini = 1001 - Somme= 1.6439345666815615
Jusqu'à quelle précision? 4
Rang mini = 10001 - Somme= 1.6448340718480652
Jusqu'à quelle précision? 5
Rang mini = 100001 - Somme= 1.6449240668982423
```

L'exécution de ce programme montre une curieuse propriété qui se devinait déjà avec les résultats précédents : le 1er terme à partir duquel $\frac{\pi^2}{6}-G_n$ dépasse 10^{-p} est 10^p+1. Cela se vérifie jusqu'à p=5 mais on peut pousser l'exploration plus loin. Pour p=6 c'est toujours vrai, pour p=7 je trouve 10000098... Comme le temps de calcul commence à être plus long, j'en resterai là avec ce programme et cette propriété.

c) ζ(3) : La constante d'Apéry
Apéry a montré (1978) que la somme des inverses des cubes d'entiers positifs, notée ζ(3), est irrationnelle.
Notons $H(n)$ la somme des n premiers inverses des cubes d'entiers positifs.
►Calculer à la main les valeurs exactes de $H(n)$, pour $n\in\{1,2,3,4,5\}$.
On a $H(1)=\frac{1}{1^3}=1$; $H(2)=\frac{1}{1^3}+\frac{1}{2^3}=\frac{9}{8}=1{,}125$; $H(3)=H(2)+\frac{1}{3^3}=\frac{9}{8}+\frac{1}{27}=\frac{251}{216}\approx 1{,}162$;
$H(4)=H(3)+\frac{1}{4^3}=\frac{2035}{1728}\approx 1.1776620370370370 4$;
$H(5)=H(4)+\frac{1}{5^3}=\frac{256103}{216000}\approx 1.1856620370370370 3$.

► Écrire un algorithme qui permet de déterminer une valeur approchée de $H(n)$. Déterminer $H(10)$.
Pour calculer une valeur approchée de $H(n)$, voici l'algorithme utilisé :

Lire le nombre n
S=0
Pour i allant de 1 à n : S=S+1/($i\times i\times i$)
Afficher S

Je programme sur mon ordinateur cet algorithme en langage Python :

```
n=int(input("Jusqu'à quelle rang? "))
S=0
for i in range(1,n+1):
    S+=1/(i*i*i)
print("Somme= {} ".format(S))
```

```
Jusqu'à quelle rang? 5
Somme= 1.185662037037037
```

Je teste ensuite pour n=5 et trouve 1.185662... ce qui me confirme l'exactitude de mon programme.

Pour $H(10)$, on trouve alors 1.197531985674193.
$H(20)=\frac{336658814638864376538323}{280346265322438720204800}\approx 1.2008678419584369 5810$ (j'ai utilisé Xcas pour trouver la valeur exacte) ;
$H(100)\approx 1.202007400659677 61040$;
$H(1000)\approx 1.202056403659344 28549$;
$H(10000)\approx 1.202056898160094 26040$ (il a fallu à l'ordinateur une demi-seconde pour ce calcul) ;
$H(100000)\approx 1.202056903109594 78541$ (il faut 37,113 secondes pour la calculer).

►Modifier l'algorithme pour que celui-ci affiche la valeur de n à partir de laquelle, du point de vue de la calculatrice, $H(n)=H(n+1)$. Déterminer cette valeur sur votre calculatrice ainsi que la valeur approchée de ζ(3) obtenue, la durée du calcul et le modèle de la calculatrice.

Comme dans le 1er problème, nous voulons nous confronter au moment où la calculatrice (ou l'ordinateur) est pris en défaut. La machine doit coder les nombres réels comme des décimaux (on dit des flottants, car ils ont une puissance de dix, en fait de deux, qui leur est attaché). L'algorithme que l'on va utiliser est donc :

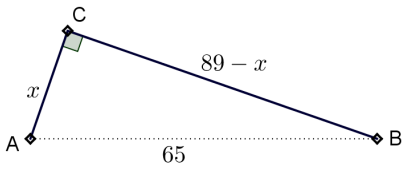
```
S1=0
S2=1
i=2
while S1<S2:
    S1=S2
    S2+=1/(i**3)
    i+=1
print("Rang égalité = {} - Somme= {} ".format(i,S1))
```

```
Rang égalité = 208065 - Somme= 1.202056903150321
```

Je garde en mémoire le terme précédent, que je mets dans la mémoire S1 et je calcule le nouveau dans la mémoire S2. Le programme s'arrête quand S1=S2.
Ce programme s'exécute rapidement et nous livre la réponse : n=208065.
Le nombre obtenu est 1.202056903150321. Donc pour nous ζ(3) ≈ 1.202056903150321.
Cela signifie que les termes $H(208065)$ et $H(208064)$ sont égaux, du moins semble l'être pour le programme. Si on calcule $H(300000)$ au moyen du 1er programme, on ne trouvera pas davantage.
En Python la limite se situe là car les nombres réels sont traités avec un certain dispositif qui doit être différent sur une calculatrice.
Sur une TI-83 Premium CE l'égalité est obtenue pour n=1010 en 23 secondes. La valeur trouvée est ζ(3) ≈ 1,202056413 (6 décimales exactes). Sur ma vieille Casio Graph25, l'égalité est obtenue pour n=23248 au bout d'un temps relativement long (environ 25 minutes). La valeur trouvée est ζ(3) ≈ 1,202056902 (8 décimales exactes). Il faut savoir qu'il y a des décimales cachées dans une calculatrice : les décimales affichées ne sont pas toute la précision calculée. Pour s'en apercevoir, on peut tester l'opération suivante :
Sur ma calculatrice, la valeur affichée est 1,202056902 à l'issue de l'exécution du programme. Je tape alors Ans – 1,202056902. Elle m'affiche alors $1{,}2^{-10}$, ce qui signifie $1{,}2\times10^{-10}$, soit 0,00000000012. Il y a donc 2 chiffres cachés sur ma calculatrice. Elle en affiche 9 mais en calcule 11.

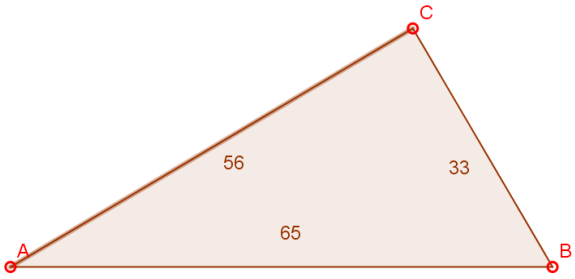
0→S	T→S	I◢
1→T	T+1/(I^3)→T	T◢
2→I	I+1→I	
While S<T	WEnd	

La vraie valeur de ζ(3) commence par 1.202056903159594285399738161511449990764986292340 49...
Notre programme Python n'a trouvé que les 11 premières décimales ; celui de la calculatrice TI-83 n'en a trouvé que les 6 premières ; celui de ma Casio Graph25 en a trouvé les 8 premières.



II] Bout de ficelle

a) Une ficelle de longueur L=89 *cm* est fixée à ses extrémités par deux clous A et B distants de e=65 *cm*. Est-il possible de tendre la ficelle de manière à ce que le triangle ABC soit rectangle en C ?



Pour que le triangle soit rectangle il faut vérifier la relation de Pythagore : $x^2+(89-x)^2=65^2$. En développant, on trouve que le petit côté AC=x doit vérifier :
$x^2+7921-188x+x^2=4225\Leftrightarrow 2x^2-188x+3696=0\Leftrightarrow x^2-89x+1848=0$.
Le discriminant $\Delta=89^2-4\times1848=529=23^2$ étant positif, cette équation a deux solutions :

$$x_1=\frac{89+23}{2}=\frac{112}{2}=56 \text{ et } x_2=\frac{89-23}{2}=\frac{66}{2}=33 .$$

On remarque que si AC= x_1 , alors BC= x_2 , et réciproquement. Il n'y a donc qu'une possibilité de triangle rectangle avec cette ficelle de 89 *cm* de long. Nous avons tracé la version AC=56, BC=33.
Ses côtés mesurent 65, 89 et 33 *cm*.

b) Les clous étant distants de 65 *cm*, quelle est la longueur maximale L_{max} de la ficelle pour que le problème soit possible ? Quelle est la longueur entière minimale L_{min} de la ficelle pour que le problème soit possible ?
Lorsqu'on change la valeur de la longueur de la ficelle, on modifie l'équation. Avec une longueur L, celle-ci devient : $x^2+(L-x)^2=65^2$, soit $x^2+L^2-2Lx+x^2=4225 \Leftrightarrow 2x^2-2Lx+L^2-4225=0$.
Le discriminant $\Delta=4L^2-4\times2\times(L^2-4225)=8\times4225-4L^2=33800-4L^2=4(8450-L^2)$ n'est positif que lorsque $4(8450-L^2)\geq0$, soit pour $L^2\leq8450$. Comme, il faut en plus avoir $L>0$ (la ficelle doit exister!), l'inéquation en L a pour solutions (on peut faire un tableau de signes pour s'en assurer) :

$$0<L\leq\sqrt{8450} \text{ c'est-à-dire } 0<L\leq91{,}92388 \text{ } cm.$$

Mais il y a d'autres conditions à vérifier :
les longueurs AC=x et BC= $L-x$ doivent aussi être positives.
Si la longueur x vaut 0, la longueur $L-x$ vaut 65.
Si la longueur x vaut L, la longueur $L-x$ vaut 0.
Entre ces deux extrêmes, il est toujours possible de trouver une valeur de x qui solutionne le problème, jusqu'à $L=\sqrt{8450}\approx91{,}92388$ *cm*.

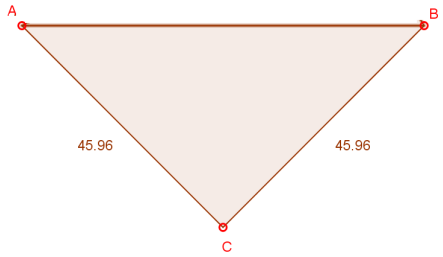
La longueur de ficelle la plus courte est 65 *cm* (certains penseront que cette valeur n'est pas acceptable car alors le triangle n'existe pas, ou du moins il est plat, les trois sommets étant alignés ; on peut cependant construire un triangle rectangle ayant 0, 65 et 65 comme côtés : il vérifie l'égalité de Pythagore $0^2+65^2=65^2$; les réticents accepteraient 65,1 *cm* ou même 65,0000000001 *cm* mais pas 65 ?) et la plus longue est $\sqrt{8450}\approx91{,}92388$ *cm*. Pour résumer cela : $65 \text{ } cm \leq L \leq \sqrt{8450}$ *cm*.
Si on décide que x est la plus petite des deux longueurs, alors :

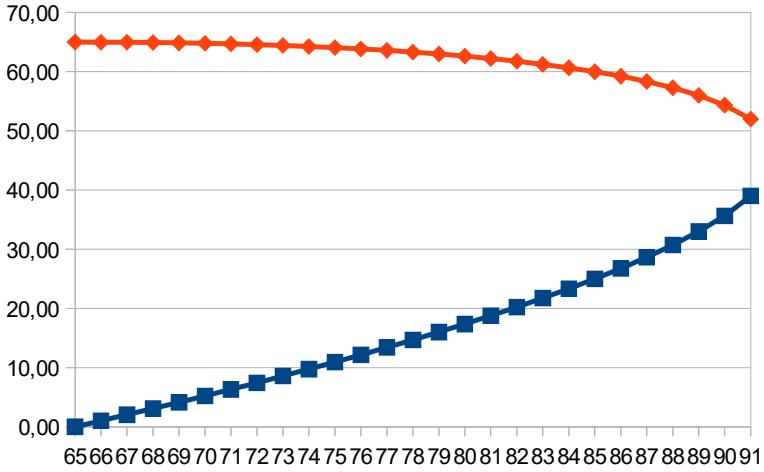
$$x=\frac{2L-\sqrt{4(8450-L^2)}}{4}=\frac{L-\sqrt{8450-L^2}}{2} \text{ et } L-x=L-\frac{L-\sqrt{8450-L^2}}{2}=\frac{L+\sqrt{8450-L^2}}{2} .$$

On peut essayer plusieurs valeurs entre 0 et $\sqrt{8450}\approx91{,}92388$ et donner la dimension des côtés :

L	65	66	70	80	89	90	91	$\sqrt{8450}$
x	0	1,01	5,21	17,36	33	35,65	39	45,96
$L-x$	65	64,99	64,79	62,64	56	54,32	52	45,96

Le graphique résume cette approche et la figure ci-dessous donne le triangle rectangle correspondant à la plus longue des ficelles acceptable : c'est un demi carré ! Pour cette longueur de $\sqrt{8450}\approx91{,}92388$ *cm*, les longueurs ACB et BC sont égales à :
$x= L-x=\frac{\sqrt{8450}}{2}=\sqrt{2112{,}5}\approx45{,}96$ *cm*.



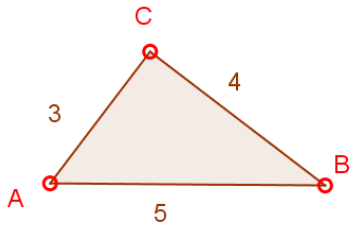


c) La ficelle a une longueur L qui dépasse de 2 *cm* l'écart e entre les deux clous (L=e+2).
À partir de quelle valeur exacte de e (mesuré en *cm*), le problème a t-il des solutions ?
Donner les solutions, pour la plus petite valeur entière de e.
Lorsque la longueur L dépasse de 2 *cm* l'écart e entre les deux clous, on modifie encore l'équation.
Avec une longueur $L=e+2$, celle-ci devient :
$x^2+(e+2-x)^2=e^2$, soit $x^2+(e+2)^2-2(e+2)x+x^2=e^2 \Leftrightarrow 2x^2-2(e+2)x+(e+2)^2-e^2=0$.
Cela s'arrange encore : $x^2-(e+2)x+2(e+1)=0$.
Le discriminant $\Delta=(e+2)^2-4\times2\times(e+1)=e^2+4e+4-8e-8=e^2-4e-4=(e-2)^2-8$ n'est positif que lorsque $(e-2)^2-8\geq0$, soit pour $(e-2)^2\geq8$.
L'inéquation en e a pour solutions (on peut faire un tableau de signes pour s'en assurer) :

$e-2\geq\sqrt{8}$ ou $e-2\leq-\sqrt{8}$, c'est-à-dire $e\geq2+\sqrt{8}$ *cm* ou $e\leq2-\sqrt{8}\approx-0{,}82$ *cm*. La dernière inégalité n'est pas acceptable car *e* est une longueur, et donc, doit être positive. Il faut donc avoir $e\geq2+\sqrt{8}\approx4{,}828$ *cm*

C'est à partir de cette valeur de $e=2+\sqrt{8}$ *cm*, que le problème a des solutions. Celles-ci s'écrivent :

$x=\frac{e+2-\sqrt{(e-2)^2-8}}{2}$ et $L-x=e+2-x=\frac{e+2+\sqrt{(e-2)^2-8}}{2}$.



La plus petite valeur entière de *e* est donc 5 *cm*.

Pour cette valeur, on trouve *x*=3 *cm* et $L-x=e+2-x=4$ *cm*.

C'est le fameux triangle 3-4-5 des égyptiens.

Donnons quelques autres valeurs :

e	$2+\sqrt{8}$	5	6	10	20	50	65	100
x	$2+\sqrt{2}\approx3{,}41$	3	2,59	2,26	2,11	2,04	2,03	2,02
$L-x$	$2+\sqrt{2}\approx3{,}41$	4	5,41	9,74	20,89	49,96	64,97	99,98

On peut remarquer que *e*=5 est la seule valeur pour laquelle *x* est entier (pour toutes les autres valeurs on est compris entre 2 exclu et 3.

d) La ficelle a une longueur L égale à 1,25 fois l'écart *e* entre les deux clous ($L=\frac{5}{4}e$).

Montrer que, pour toutes les valeurs non-nulles de *e*, le problème a toujours deux solutions.

Exprimer alors les côtés des triangles solution en fonction de *e*.

Y a t-il des solutions avec des côtés entiers ?

Lorsque la longueur *L* égale à 1,25 fois l'écart *e* entre les deux clous, on modifie l'équation.

Avec une longueur *L*=1,25*e*, celle-ci devient :

$x^2+(\frac{5}{4}e-x)^2=e^2$, soit $x^2+\frac{25}{16}e^2-\frac{5}{2}ex+x^2=e^2\Leftrightarrow2x^2-\frac{5}{2}ex+(\frac{25}{16}-1)e^2=0\Leftrightarrow32x^2-40ex+9e^2=0$.

Le discriminant $\Delta=(40e)^2-4\times32\times9e^2=(1600-1152)e^2=7(8e)^2$ est toujours positif.

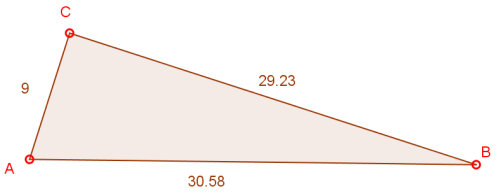
L'inéquation en *e* a donc toujours deux solutions. Celles-ci s'écrivent :

$x=\frac{(40-8\sqrt{7})e}{64}=\frac{(5-\sqrt{7})e}{8}$ et $L-x=\frac{5}{4}e-x=\frac{(5+\sqrt{7})e}{8}$.

Y a t-il des solutions avec des côtés entiers ? Cela n'est pas possible, car si *e* est entier, alors $\frac{(5-\sqrt{7})e}{8}$ est le huitième d'un multiple de $5-\sqrt{7}$ qui est irrationnel. Pour que $\frac{(5-\sqrt{7})e}{8}$ soit entier, il faudrait que *e* soit un multiple de $5+\sqrt{7}$, car alors $(5-\sqrt{7})(5+\sqrt{7})=25-7=18$.

Par exemple, avec $e=4(5+\sqrt{7})\approx30{,}58$,

on aurait $x=\frac{(5-\sqrt{7})e}{8}=9$. Il y aurait un des côtés entier, et les deux autres irrationnels (voir figure).



Donnons quelques autres valeurs :

e	1	2	5	10	20	$4(5+\sqrt{7})$	65	100
x	0,29	0,59	1,47	2,94	5,89	9	19,13	29,43
$L-x$	0,96	1,91	4,78	9,56	19,11	29,23	62,12	95,57